



AKADEMIA GÓRNICZO-HUTNICZA

Im. Stanisława Staszica w Krakowie



Wydział Informatyki, Elektroniki i Telekomunikacji

Dokumentacja projektu

Configuration and testing of HA-enabled Postgres-XL Cluster

Konfiguracja i testowanie klastra Postgres-XL
z włączoną obsługą wysokiej dostępności

Kierunek: Informatyka

Rok: IV

Autor: Patryk Małek, Jakub Kolasiak

Spis treści

Spis treści	2
1. Wprowadzenie	3
1.1. Replikacja geograficzna	3
1.2. Skalowanie	3
1.3. Słownik pojęć	3
2. Środowisko uruchomieniowe/testowe	4
3. Instalacja Postgres-XL na węzłach klastra	8
4. Etapy konfiguracji i testowanie	15
4.1. Konfiguracja klastra	15
4.2. Zainicjowanie klastra:	17
4.3. Uruchomienie klastra:	18
4.4 Testowanie klastra:	26
4.5 Monitorowanie klastra:	28

1. Wprowadzenie

Celem dokumentacji jest opis konfiguracji i testowania klastra Postgres-XL z włączoną obsługą wysokiej dostępności (HA) na potrzeby projektu na laboratorium Administracji Systemami Komputerowymi.

Klaster ten będzie oparty na PostgreSQL, umożliwiając skalowanie pionowe i poziome.

1.1.Replikacja geograficzna

Replikacja geograficzna może być realizowana poprzez replikację logów (*log shipping*).

Replikacja odbywa się między węzłami bazy danych. Cały proces polega na przesłaniu do innego węzła wszystkich zapisanych transakcji.

Kluczowymi czynnikami wpływającymi na skuteczność tej metody jest stabilne oraz szybkie łącze między maszynami oraz jak najniższe opóźnienia. Wysokich opóźnień mogą prowadzić do problemów w poprawnej synchronizacji danych.

1.2.Skalowanie

Skalowanie pionowe polega na zwiększeniu mocy obliczeniowej jednego węzła poprzez dodanie pamięci RAM, zwiększenie liczby procesorów lub wymianę na mocniejszy sprzęt.

Skalowanie poziome polega na dodawaniu kolejnych węzłów do klastra w celu zwiększenia jego mocy obliczeniowej i pojemności.

Oba rodzaje skalowania można wykorzystać w Postgres-XL w celu poprawy wydajności i odporności systemu na awarie.

1.3.Słownik pojęć

AZ - (ang. *Availability Zone*) - grupa węzłów bazy danych znajdujących się w jednym regionie geograficznym, ale umieszczonych w różnych strefach dostępności w ramach tego regionu. Dzięki AZ możemy zwiększyć niezawodność i odporność na awarie (w przypadku awarii pozostałe węzły AZ mogą przejąć zadania uszkodzonej jednostki).

RDS - (ang. *Relational Database Service*) - usługa systemu do zarządzania relacyjnymi bazami danych w AWS. Umożliwia ona łatwe tworzenie, skalowanie i zarządzanie bazami danych.

MPLS - (ang. *Multiprotocol Label Switching*) - technologia w zapewniająca szybki i niezawodny przesył danych poprzez zapewnienie odpowiedniego priorytetu i przepływności ruchu sieciowego.

2. Środowisko uruchomieniowe/testowe

W celu uruchomienia klastra Postgres-XL w środowisku testowym wykorzystaliśmy Vagrant oraz Virtualbox.

Vagrant to narzędzie do automatycznego zarządzania wirtualnymi maszynami, które pozwala na prostą i powtarzalną konfigurację środowiska.

Virtualbox to darmowe oprogramowanie do wirtualizacji systemów operacyjnych, które umożliwia uruchomienie wirtualnych maszyn na komputerze.

Wersja środowiska:

- Vagrant 2.3.6
- Virtualbox 7.0.8 r156879
- debian/bullseye64 (Vagrant Box)
- PostgreSQL 11

W celu uruchomienia środowiska podjęliśmy kroki:

1. Pobrano i zainstalowano VirtualBox oraz Vagrant z oficjalnych stron internetowych dla systemu Windows 10.
2. Uruchomiono VirtualBox i sprawdzono, czy działa poprawnie.
3. Po zainstalowaniu Vagrant, należało dostosować wersję VirtualBox Guest Additions na wirtualnych maszynach do wersji VirtualBox komendą:

vagrant plugin install vagrant-vbguest

4. Zainicjowano projekt Vagrant w wybranym katalogu za pomocą polecenia:

vagrant init debian/bullseye64

```
Microsoft Windows [Version 10.0.19045.2846]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

D:\Coding Studio\Semestr VIII\ASK\posgres-xl-cluster>vagrant init debian/bullseye64
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

5. Następnie zmodyfikowano konfigurację pliku Vagrantfile w celu stworzenia automatycznie 5 wirtualnych maszyn oraz automatycznej instalacji PostgreSQL za pomocą skryptu w pliku postgres-install.sh:

```
Vagrantfile X
Vagrantfile
1  # -*- mode: ruby -*-
2  # vi: set ft=ruby :
3
4  Vagrant.configure("2") do |config|
5    # Konfiguracja 4 wirtualnych maszyn na potrzeby Laboratorium AGH ASK
6    config.vm.define "node1" do |node1|
7      node1.vm.box = "debian/bullseye64"
8      node1.vm.hostname = "node1"
9      node1.vm.network "private_network", ip: "10.0.0.101"
10     node1.vm.network "forwarded_port", guest: 5432, host: 54321, id: "postgresql-1"
11     node1.vm.network "forwarded_port", guest: 22, host: 2201, auto_correct: true, id: "ssh"
12     node1.vm.provider "virtualbox" do |vb|
13       vb.memory = "1024"
14     end
15     # Skrypt instalujący PostgreSQL
16     node1.vm.provision "shell", path: "postgres-install.sh"
17   end
18
19   config.vm.define "node2" do |node2|
20     node2.vm.box = "debian/bullseye64"
21     node2.vm.hostname = "node2"
22     node2.vm.network "private_network", ip: "10.0.0.102"
23     node2.vm.network "forwarded_port", guest: 5432, host: 54322, id: "postgresql-2"
24     node2.vm.network "forwarded_port", guest: 22, host: 2202, auto_correct: true, id: "ssh"
25     node2.vm.provider "virtualbox" do |vb|
26       vb.memory = "1024"
27     end
28     # Skrypt instalujący PostgreSQL
29     node2.vm.provision "shell", path: "postgres-install.sh"
30   end
31
32   config.vm.define "node3" do |node3|
33     node3.vm.box = "debian/bullseye64"
34     node3.vm.hostname = "node3"
35     node3.vm.network "private_network", ip: "10.0.0.103"
36     node3.vm.network "forwarded_port", guest: 5432, host: 54323, id: "postgresql-3"
37     node3.vm.network "forwarded_port", guest: 22, host: 2203, auto_correct: true, id: "ssh"
38     node3.vm.provider "virtualbox" do |vb|
39       vb.memory = "1024"
40     end
41     # Skrypt instalujący PostgreSQL
42     node3.vm.provision "shell", path: "postgres-install.sh"
43   end
44
45   config.vm.define "node4" do |node4|
46     node4.vm.box = "debian/bullseye64"
47     node4.vm.hostname = "node4"
48     node4.vm.network "private_network", ip: "10.0.0.104"
49     node4.vm.network "forwarded_port", guest: 5432, host: 54324, id: "postgresql-4"
50     node4.vm.network "forwarded_port", guest: 22, host: 2204, auto_correct: true, id: "ssh"
51     node4.vm.provider "virtualbox" do |vb|
52       vb.memory = "1024"
53     end
54     # Skrypt instalujący PostgreSQL
55     node4.vm.provision "shell", path: "postgres-install.sh"
56   end
57
58   config.vm.define "node5" do |node5|
59     node5.vm.box = "debian/bullseye64"
60     node5.vm.hostname = "node5"
61     node5.vm.network "private_network", ip: "10.0.0.105"
62     node5.vm.network "forwarded_port", guest: 22, host: 2205, auto_correct: true, id: "ssh"
63     node5.vm.provider "virtualbox" do |vb|
64       vb.memory = "1024"
65     end
66     # Skrypt instalujący PostgreSQL
67     node5.vm.provision "shell", path: "postgres-install.sh"
68   end
69 end
```

6. Zawartość pliku postgres-install.sh:

```
postgres-install.sh
1  #!/bin/bash
2
3  # Aktualizacja systemu
4  apt-get update
5  apt-get upgrade -y
6
7  # Instalacja PostgreSQL na maszynie
8  apt-get install -y postgresql postgresql-contrib
9  systemctl enable postgresql.service
10 systemctl start postgresql.service
11
12 # Ustawienie hasła dla użytkownika postgres
13 sudo -u postgres psql -c "ALTER USER postgres PASSWORD 'agh_lab';"
14
15 systemctl stop postgresql.service
16 # Ustawienie hasła użytkownika systemowego postgres
17 echo "postgres:agh_lab" | sudo chpasswd
18
19 # Ustawienie katalogu domowego użytkownika postgres na /home/postgres
20 sudo usermod --home /home/postgres postgres
21
22 # Dodanie użytkownika postgres do grupy sudo
23 sudo usermod -aG sudo postgres
24
25 # Konfiguracja SSH dla użytkownika postgres
26 sudo sed -i 's/^PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
27 sudo systemctl restart sshd.service
28
29 # Konfiguracja PostgreSQL
30 # Zezwolenie w pliku pg_hba na to że dowolny użytkownik z dowolnego adresu IP może łączyć
31 echo "host all all 0.0.0.0/0 md5" >> /etc/postgresql/11/main/pg_hba.conf
32 # Zezwolenie na nasłuchiwanie połączeń na wszystkich adresach IP
33 echo "listen_addresses='*'" >> /etc/postgresql/11/main/postgresql.conf
34 # Restart po zmianach
35 systemctl restart postgresql.service
36
```

7. Uruchomiono konfigurację za pomocą polecenia:

`vagrant up`

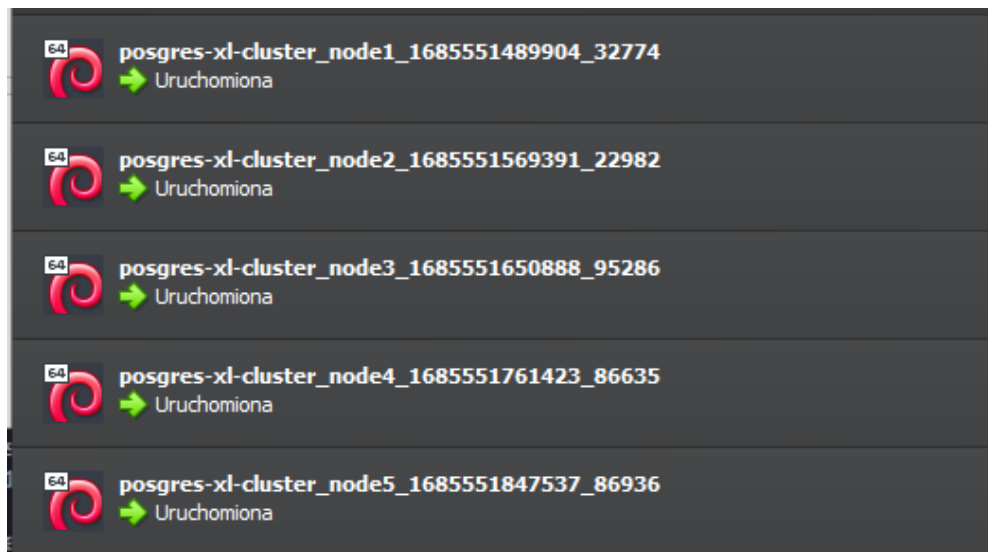
8. Maszyny zostały utworzone:

```
==> node1: Machine 'node1' has a post `vagrant up` message. This is a message
==> node1: from the creator of the Vagrantfile, and not from Vagrant itself:
==> node1:
==> node1: Vanilla Debian box. See https://app.vagrantup.com/debian for help and bug reports

==> node2: Machine 'node2' has a post `vagrant up` message. This is a message
==> node2: from the creator of the Vagrantfile, and not from Vagrant itself:
==> node2:
==> node2: Vanilla Debian box. See https://app.vagrantup.com/debian for help and bug reports

==> node3: Machine 'node3' has a post `vagrant up` message. This is a message
==> node3: from the creator of the Vagrantfile, and not from Vagrant itself:
==> node3:
==> node3: Vanilla Debian box. See https://app.vagrantup.com/debian for help and bug reports

==> node4: Machine 'node4' has a post `vagrant up` message. This is a message
==> node4: from the creator of the Vagrantfile, and not from Vagrant itself:
==> node4:
==> node4: Vanilla Debian box. See https://app.vagrantup.com/debian for help and bug reports
```



9. Sprawdzono poprawność uruchomienia systemu maszyny wirtualnej poprzez zalogowanie się na maszynę wykorzystując ssh:

```
D:\Coding Studia\Semestr VIII\ASK\posgres-xl-cluster>vagrant ssh node1
Linux node1 5.10.0-20-amd64 #1 SMP Debian 5.10.158-2 (2022-12-13) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Apr 14 18:53:36 2023 from 10.0.2.2
vagrant@node1:~$
```

10. Poleceniem *ping* sprawdzono poprawną komunikację z innymi maszynami:

```
vagrant@node1:/$ ip a show eth1 | grep -v inet6 | grep -v valid_lft
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:68:85:45 brd ff:ff:ff:ff:ff:ff
    altname enp0s8
    inet 10.0.0.101/24 brd 10.0.0.255 scope global eth1
vagrant@node1:/$ ping 10.0.0.102
PING 10.0.0.102 (10.0.0.102) 56(84) bytes of data.
64 bytes from 10.0.0.102: icmp_seq=1 ttl=64 time=0.709 ms
^C
--- 10.0.0.102 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.709/0.709/0.709/0.000 ms
vagrant@node1:/$ ping 10.0.0.103
PING 10.0.0.103 (10.0.0.103) 56(84) bytes of data.
64 bytes from 10.0.0.103: icmp_seq=1 ttl=64 time=0.682 ms
^C
--- 10.0.0.103 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.682/0.682/0.682/0.000 ms
vagrant@node1:/$ ping 10.0.0.104
PING 10.0.0.104 (10.0.0.104) 56(84) bytes of data.
64 bytes from 10.0.0.104: icmp_seq=1 ttl=64 time=0.922 ms
^C
--- 10.0.0.104 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.922/0.922/0.922/0.000 ms
vagrant@node1:/$
```

Specyfikacja dla każdej z utworzonych maszyn wirtualnych:

- Nazwa maszyny: nodeX
- System operacyjny: Debian Bullseye (64-bit)
- Adres IP: 10.0.0.10X
- Nazwa sieci wirtualnej: private_network
- Pamięć RAM: 1024 MB

Gdzie X to numer kolejny maszyny, która będzie stanowiła węzeł klastra Postgres-XL

3. Instalacja Postgres-XL na węzłach klastra

Klaster to zbiór węzłów bazy danych, które pracują razem, aby dostarczać usługi bazy danych.

Węzeł bazy danych to serwer lub instancja bazy danych, która jest częścią klastra Postgres-XL i która pełni określoną rolę w klastrze. W klastrze Postgres-XL istnieją trzy rodzaje węzłów:

1. Węzeł koordynatora (ang. coordinator node) - jest to węzeł, który jest odpowiedzialny za koordynowanie działania klastra i dystrybuowanie zapytań między węzłami datanode. W klastrze Postgres-XL może być tylko jeden węzeł koordynatora.
2. Węzeł datanode (ang. datanode) - jest to węzeł, na którym przechowywane są dane bazy danych. W klastrze Postgres-XL może być wiele węzłów datanode.

3. GTM (ang. Global Transaction Manager) - jest to węzeł, który zarządza transakcjami w klastrze i zapewnia spójność danych między węzłami datanode.

Według [dokumentacji](#) należy zapewnić między innymi “password-less ssh access”, tak, aby każda maszyna miała dostęp do siebie nawzajem bez podawania hasła za pomocą ssh:

A few pre-requisites are necessary on each node that is going to be a part of the Postgres-XL setup.

- Password-less ssh access is required from the node that is going to run the pgxc_ctl utility.
- The PATH environment variable should have the correct Postgres-XL binaries on all nodes, especially while running a command via ssh.
- The pg_hba.conf entries must be updated to allow remote access. Variables like coordPgHbaEntries and datanodePgHbaEntries in the pgxc_ctl.conf configuration file may need appropriate changes.
- Firewalls and iptables may need to be updated to allow access to ports.

Za pomocą następującej komendy, należy wygenerować parę kluczy na każdej z maszyn.

```
vagrant@node1:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:JiPQi1wmY+bVITnImITZPJcek5LyY1MozSR+dxEtvMA vagrant@node1
The key's randomart image is:
+---[RSA 3072]-----+
|
|  =+*+0..oo
|  o+0B^..E o..
|  * 0X+= ... o
|  = 0 o.. ..
|  + o o S
|  . +
|
+-----[SHA256]-----+
vagrant@node1:~$
```

Tak wygenerowany klucz publiczny należy przesłać na każdą z maszyn, dodając do pliku “authorized_keys” na drugiej maszynie. Czynność należy również powtórzyć dla każdej z maszyn.

```
vagrant@node1:~/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDHH9FkNcg4vAqLVDIumbLLZKc4h+UCtBoUy3rHCv
wkPTw0xIBONUNZ+YftM2xHiv59cIdkHIDUmILCJ5XMPA92fZrrXzmEw5j681utk01DUsmvh18FTUymF
Nq/y9+kQjPKp9T4Cw7bSTZv9FTda/Kr3 vagrant
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDUn2emhUgIFW5mmIeZF/9akvocOFSLoaIoHTvOesv
zqRgi/MUAX7W7ZUbr/An8R+XwrKXig0JJvHRNw9Uz7awxxosuxwdVcQC3tBct4FM0dadXw3IlyvX4g
vw0wMM25AmeImFTKZ7Hqi00XM2y1wzWiRXhBjRS2K5pTYhWKNmN+Pn+J/TRUM8/gnsNNLPHPkI+fiGI
wv8v1kF8U9xZmriRPh7hQRmPEQYnc= vagrant@node2
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDQDLhY9Ne2AMwvhvIqz4YHPNJ2sLS0JXVtonhRynmrc
Rji5U/HITqNaey0t8iRedGFfgnn+XGUL4f4u5TAjLS9SLycdKPBm4Ns+vgv6LS49qxbSfnCcNcrTyVt
/30vxAALIZJUD5ouJmB8l8INYvXo7kyJ03i1oQgXGUyJLWza/0noxS2u8bEF0nhtvWftcMVKwRBJY9NTE
E+3hYVRUu8Zwg0gpe0PChz8ACxzGc= vagrant@node3
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDQDN+ojh5UiSh5WfncME0kZaIoJBB/aNhmpwU4MELI6
jj0usHCTLbCc6MltomZoZK2Gw6yWG6w6UOYlftwNuuwrDzkofUVjIcqFidzrGCHnepB8PE4IjahntL+
Mgqhzi0FFLmhDgtXbZ+eHXK5U7ppqAC6VbIkRzeF+EJKDmSvohgU5dwYsS9f20FauGTnsIGZ0vptND
ZuLNF0xsVHgtA83laHndgU4nF4qpc= vagrant@node4
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDQDiYPhBM6Jd1KJp4nSgasl5hFajmQKR+cpw0I2c0XQ
qz1PhhZETuX+EXsmRdQLMmF8IkuqRdMBA0p7mAcE3jqUjuweF9FQVgx2KyIuOuuAncOy53b25mQfhq3
JNoEU4F11avGIefz5FUdoMYnz8+6wZNF6raW+1o6RSQVgYU5SR65/KPebTAFEz690K7DJ0MbZaOKUT
3dkKpAzqjy+JSYHjuAc+/tPo4vPSM= vagrant@gtm-node
vagrant@node1:~/.ssh$
```

Aby zautomatyzować tą operację można użyć do tego celu narzędzia Ansible:

```
ansible > playbooks > ssh_distribution.yml > {} 0 > [ ] tasks > {} 3 > [ ] with_items
1  ---
2  - name: Exchange Keys between servers
3    remote_user: postgres
4    hosts: target_hosts
5    tasks:
6      - name: Usuń plik .ssh/authenticated
7        file:
8          path: ~/.ssh/
9          state: absent
10
11      - name: SSH KeyGen command
12        tags: run
13        shell: >
14          ssh-keygen -q -b 2048 -t rsa -N "" -f ~/.ssh/id_rsa
15
16      - name: Fetch the keyfile from one server to another
17        tags: run
18        fetch:
19          src: "~/.ssh/id_rsa.pub"
20          dest: "buffer/{{ansible_hostname}}-id_rsa.pub"
21          flat: yes
22
23      - name: Skopiuj klucz do pliku authorized_keys przy użyciu modułu Ansible
24        tags: run
25        authorized_key:
26          user: postgres
27          state: present
28          key: "{{ lookup('file', 'buffer/{{item.dest}}-id_rsa.pub') }}"
29        when: item.dest != ansible_hostname
30        with_items:
31          - { dest: "{{groups['target_hosts'][0]}}" }
32          - { dest: "{{groups['target_hosts'][1]}}" }
33          - { dest: "{{groups['target_hosts'][2]}}" }
34          - { dest: "{{groups['target_hosts'][3]}}" }
35          - { dest: "{{groups['target_hosts'][4]}}" }
```

Operacja przebiegła pomyślnie, z każdej maszyny można dostać się na inną bez hasła, jedynie za pomocą klucza publicznego ssh.

```
vagrant@node1:~/.ssh$ ssh 10.0.0.102
Linux node2 5.10.0-23-amd64 #1 SMP Debian 5.10.179-1 (2023-05-12) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon May 15 16:38:02 2023 from 10.0.2.2
vagrant@node2:~$ exit
```

Z oficjalnej strony pobraliśmy paczkę postgres-xl w wersji 10r1.1. Taki plik należy rozpakować:

```
vagrant@node1:~$ ls
postgres-xl-10r1.1.tar.gz
vagrant@node1:~$ tar zxvf postgres-xl-10r1.1.tar.gz
postgres-xl-10r1.1/
postgres-xl-10r1.1/.dir-locals.el
postgres-xl-10r1.1/INSTALL
postgres-xl-10r1.1/GNUMakefile.in
postgres-xl-10r1.1/configure
```

```
vagrant@node1:~/postgres-xl$ ls
COPYRIGHT      HISTORY  LICENSE.txt  README      config      configure.in  doc
GNUMakefile.in INSTALL  Makefile     aclocal.m4  configure   contrib      src
vagrant@node1:~/postgres-xl$ ./configure
```

Trzeba również zainstalować narzędzia potrzebne w późniejszym procesie kompilacji i instalacji:

```
vagrant@node1:~/postgres-xl$ sudo apt-get install build-essential tar libreadline-dev zlib1g-dev libssl-dev libxml2-dev flex
```

Poleceniem “./configure”, przygotowanie plików konfiguracyjnych:

```
vagrant@node1:~/postgres-xl$ ./configure
```

Przygotowanie instalacji:

```
vagrant@node1:~/postgres-xl$ make
```

```
make[2]: Leaving directory '/home/vagrant/postgres-xl/src/test/isolation'
make -C test/perl all
make[2]: Entering directory '/home/vagrant/postgres-xl/src/test/perl'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/vagrant/postgres-xl/src/test/perl'
make[1]: Leaving directory '/home/vagrant/postgres-xl/src'
make -C config all
make[1]: Entering directory '/home/vagrant/postgres-xl/config'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/vagrant/postgres-xl/config'
All of Postgres-XL successfully made. Ready to install.
vagrant@node1:~/postgres-xl$
```

Instalacja:

```
vagrant@node1:~/postgres-xl$ sudo make install
```

```
/usr/bin/install -c -m 644 ./nls-global.mk '/usr/local/pgsql/lib/pgxs/src/nls-global.mk'
make[1]: Leaving directory '/home/vagrant/postgres-xl/src'
make -C config install
make[1]: Entering directory '/home/vagrant/postgres-xl/config'
/usr/bin/mkdir -p '/usr/local/pgsql/lib/pgxs/config'
/usr/bin/install -c -m 755 ./install-sh '/usr/local/pgsql/lib/pgxs/config/install-sh'
/usr/bin/install -c -m 755 ./missing '/usr/local/pgsql/lib/pgxs/config/missing'
make[1]: Leaving directory '/home/vagrant/postgres-xl/config'
Postgres-XL installation complete.
```

Dodanie do zmiennej PATH lokalizacji z zainstalowanym Postgres-XL.

```
vagrant@node1:~$ nano ~/.bashrc
```

```
GNU nano 3.2

# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

export PATH=$PATH:/usr/local/pgsql/bin/
```

Również dodanie do zmiennej globalnej ścieżki, która będzie zawierała dane każdego węzła.

```
vagrant@node1:~$ export dataDirRoot=$HOME/DATA/pgxl/nodes
vagrant@node1:~$ mkdir $HOME/pgxc_ctl
```

Podczas konfiguracji klastra napotkaliśmy wiele problemów, więc aby ułatwić sobie ponowne konfiguracje na wielu maszynach, zdecydowaliśmy się na wykorzystanie Ansible do automatycznej konfiguracji środowiska na wszystkich hostach jednocześnie.

Powyższe opisane kroki można wykonać za pomocą “playbook’ów” Ansible:

Instalacja potrzebnych narzędzi, kopiowanie na docelowe hosty paczki Postgres-XL oraz rozpakowywanie:

```
Ansible Playbook - Ansible playbook files (ansible.json)
---
- name: Install required packages and copy postgres package to remote hosts and install postgres-xl
  hosts: target_hosts
  remote_user: postgres
  tasks:
    - name: Install required packages
      become: true
      remote_user: postgres
      become_method: sudo
      apt:
        name:
          - build-essential
          - tar
          - libreadline-dev
          - zlib1g-dev
          - libssl-dev
          - libxml2-dev
          - flex
        state: present

    - name: Copy file to remote host
      copy:
        src: postgres-xl-10r1.1.tar.gz
        dest: /home/postgres/

    - name: Extract file
      unarchive:
        src: /home/postgres/postgres-xl-10r1.1.tar.gz
        dest: /home/postgres/postgres-xl
        remote_src: yes
        extra_opts: "--strip-components=1"
```

```
pmulick@DESKTOP-8CDEJLR:/mnt/d/Coding Studio/Semestr VIII/ASK/postgres-xl-cluster/ansible$ ansible-playbook -i inventory/hosts playbooks/playbook.yml --ask-become-pass
BECOME password:

PLAY [Install required packages and copy postgres package to remote hosts] *****

TASK [Gathering Facts] *****
ok: [10.0.0.104]
ok: [10.0.0.102]
ok: [10.0.0.101]
ok: [10.0.0.103]
ok: [10.0.0.105]

TASK [Install required packages] *****
ok: [10.0.0.102]
ok: [10.0.0.101]
ok: [10.0.0.103]
ok: [10.0.0.104]
ok: [10.0.0.105]

TASK [Copy file to remote host] *****
changed: [10.0.0.103]
changed: [10.0.0.101]
changed: [10.0.0.102]
changed: [10.0.0.105]
changed: [10.0.0.104]

TASK [Extract file] *****
changed: [10.0.0.105]
changed: [10.0.0.102]
changed: [10.0.0.104]
changed: [10.0.0.103]
changed: [10.0.0.101]
```

Konfiguracja, kompilacja i instalacja Postgres-XL za pomocą Ansible:

```
1  ---
2  - name: Install postgres-xl
3    hosts: target_hosts
4    remote_user: postgres
5    tasks:
6      - name: Run './configure' command
7        shell: ./configure
8        args:
9          chdir: /home/postgres/postgres-xl
10
11      - name: Run 'make' command
12        shell: make
13        args:
14          chdir: /home/postgres/postgres-xl
15
16      - name: Run 'sudo make install' command
17        shell: make install
18        become: true
19        remote_user: postgres
20        become_method: sudo
21        args:
22          chdir: /home/postgres/postgres-xl
```

pmalek@DESKTOP-8CDE3LR: /mnt/d/Coding Studio/Semestr VIII/ASK/posgres-xl-cluster/ansible\$ ansible-playbook -i inventory/hosts playbooks/playbook.yml --ask-become-pass

```
PLAY [Install required packages and copy postgres package to remote hosts and install postgres-xl] *****

TASK [Gathering Facts] *****
ok: [10.0.0.104]
ok: [10.0.0.102]
ok: [10.0.0.101]
ok: [10.0.0.103]
ok: [10.0.0.105]

TASK [Run './configure' command] *****
changed: [10.0.0.101]
changed: [10.0.0.102]
changed: [10.0.0.103]
changed: [10.0.0.104]
changed: [10.0.0.105]

TASK [Run 'make' command] *****
changed: [10.0.0.101]
changed: [10.0.0.102]
changed: [10.0.0.104]
changed: [10.0.0.103]
changed: [10.0.0.105]

TASK [Run 'sudo make install' command] *****
changed: [10.0.0.102]
changed: [10.0.0.101]
changed: [10.0.0.104]
changed: [10.0.0.103]
changed: [10.0.0.105]

PLAY RECAP *****
10.0.0.101      : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
10.0.0.102      : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
10.0.0.103      : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
10.0.0.104      : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
10.0.0.105      : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Ustawienie dodatkowej zmiennej LD_LIBRARY_PATH wymaganej do utworzenia klastra:

```
Ansible Playbook - Ansible playbook files (ansible.json)
1 ---
2 - name: Update .profile
3   hosts: target_hosts
4   remote_user: postgres
5   tasks:
6     - name: Dodaj linijkę do .profile
7       lineinfile:
8         path: ~/.profile
9         line: 'export LD_LIBRARY_PATH="/usr/local/pgsql/lib"'
10        insertafter: EOF
11
```

4. Etapy konfiguracji i testowanie

4.1. Konfiguracja klastra

Uruchomienie narzędzia “pgxc_ctl” i stworzenie poleceniem “prepare config minimal” pliku konfiguracyjnego dla naszego klastra Postgres-XL.

```
postgres@node1:/home/vagrant$ pgxc_ctl
/usr/local/pgsql/bin/bash
Installing pgxc_ctl_bash script as /home/postgres/pgxc_ctl/pgxc_ctl_bash.
Installing pgxc_ctl_bash script as /home/postgres/pgxc_ctl/pgxc_ctl_bash.
Reading configuration using /home/postgres/pgxc_ctl/pgxc_ctl_bash --home /home/postgres/pgxc_ctl --configuration /home/postgres/pgxc_ctl/pgxc_ctl.conf
Finished reading configuration.
***** PGXC_CTL START *****

Current directory: /home/postgres/pgxc_ctl
PGXC prepare config minimal
```

Konfiguracja w pliku “pgxc_ctl.conf”.

W pliku tym należy poprawnie ustawić adresy ip inne ustawienia dotyczące węzłów.

Poniżej konfiguracja oparta o 1 GTM, 1 Coordinator Master, 3 Datanode, na 5 różnych maszynach wirtualnych.

```
#---- OVERALL -----
#
pgxcOwner=postgres # owner of the Postgres-XC database cluster. Here, we use this
                    # both as linux user and database user. This must be
                    # the super user of each coordinator and datanode.
pgxcUser=$pgxcOwner # OS user of Postgres-XC owner

tmpDir=/tmp # temporary dir used in XC servers
localTmpDir=$tmpDir # temporary dir used here locally

configBackup=n # If you want config file backup, specify y to this value.
configBackupHost=pgxc-linker # host to backup config file
configBackupDir=$HOME/pgxc # Backup directory
configBackupFile=pgxc_ctl.bak # Backup file name --> Need to synchronize when original changed.

dataDirRoot=$HOME/DATA/pgxl/nodes
```

Konfiguracja GTM - Global Transaction Manager:

```
#---- GTM -----

# GTM is mandatory. You must have at Least (and only) one GTM master in your Postgres-XC cluster.
# If GTM crashes and you need to reconfigure it, you can do it by pgxc_update_gtm command to update
# GTM master with others. Of course, we provide pgxc_remove_gtm command to remove it. This command
# will not stop the current GTM. It is up to the operator.

#---- Overall -----
gtmName=gtm

#---- GTM Master -----

#---- Overall ----
gtmMasterServer=node1
gtmMasterPort=20001
gtmMasterDir=$dataDirRoot/gtm

#---- Configuration ---
gtmExtraConfig=none           # Will be added gtm.conf for both Master and Slave (done at initialization only)
gtmMasterSpecificExtraConfig=none # Will be added to Master's gtm.conf (done at initialization only)
```

Konfiguracja Koordynatora typu Master:

```
#---- Coordinators -----

#---- shortcuts -----
coordMasterDir=$dataDirRoot/coord_master1
# coordSlaveDir=$HOME/coord_slave
coordArchLogDir=$HOME/coord_archlog

#---- Overall -----
coordNames=coord1           # Master and slave use the same name
coordPorts=30001            # Master server listening ports
poolerPorts=30011           # Master pooler ports
# coordPgHbaEntries=(::1/128) # Assumes that all the coordinator (master/slave) accepts
#                               # the same connection
#                               # This entry allows only $pgxcOwner to connect.
#                               # If you'd like to setup another connection, you should
#                               # supply these entries through files specified below.
# coordPgHbaEntries=(127.0.0.1/32) # Same as above but for IPv4 connections
coordPgHbaEntries=(0.0.0.0/0)

#---- Master -----
coordMasterServers=node2     # none means this master is not available
coordMasterDirs=$coordMasterDir
coordMaxWALsender=5 # max_wal_senders: needed to configure slave. If zero value is specified,
# it is expected to supply this parameter explicitly by external files
# specified in the following. If you don't configure slaves, leave this value to zero.
coordMaxWALSenders=$coordMaxWALsender
# max_wal_senders configuration for each coordinator.
```


Konfiguracja trzech Datanodes jako Master:

```
#--- Datanodes -----
#--- Shortcuts -----
datanodeMasterDir=$dataDirRoot/dn_master
datanodeSlaveDir=$dataDirRoot/dn_slave
datanodeArchLogDir=$dataDirRoot/datanode_archlog

#--- Overall -----
primaryDatanode=datanode_1           # Primary Node.
datanodeNames=(datanode_1 datanode_2 datanode_3)
datanodePorts=(40001 40002 40003)    # Master and slave use the same port!
datanodePoolerPorts=(40011 40012 40013) # Master and slave use the same port!
# datanodePgHbaEntries=(::1/128)      # Assumes that all the coordinator (master/slave) accepts
#                                     # the same connection
#                                     # This list sets up pg_hba.conf for $pgxcOwner user.
#                                     # If you'd like to setup other entries, supply them
#                                     # through extra configuration files specified below.
# datanodePgHbaEntries=(127.0.0.1/32)  # Same as above but for IPv4 connections
datanodePgHbaEntries=(0.0.0.0/0)      # Same as above but for IPv4 connections

#--- Master -----
datanodeMasterServers=(node3 node4 node5) # none means this master is not available.
#                                     # This means that there should be the master but is down.
#                                     # The cluster is not operational until the master is
#                                     # recovered and ready to run.

datanodeMasterDirs=($datanodeMasterDir.1 $datanodeMasterDir.2 $datanodeMasterDir.3)
datanodeMaxWalSender=3                # max_wal_senders: needed to configure slave. If zero value is
#                                     # specified, it is expected this parameter is explicitly supplied
#                                     # by external configuration files.
#                                     # If you don't configure slaves, leave this value zero.

datanodeMaxWalSenders=($datanodeMaxWalSender $datanodeMaxWalSender $datanodeMaxWalSender)
# max_wal_senders configuration for each datanode
```

4.2. Zainicjowanie klastra:

```
root@postgres01-6-pgpc-it1 init all
for i in $(cat /dev/urandom | fold -w 8 | uniq); do
    Installing pgsql-bash script as /home/postgres/pgpc-it1/pgpc-it1_bash.
    Installing pgsql-bash script as /home/postgres/pgpc-it1/pgpc-it1_bash.
    Creating configuration using /home/postgres/pgpc-it1/pgpc-it1_bash --home /home/postgres/pgpc-it1 --configuration /home/postgres/pgpc-it1/pgpc-it1.conf
    Finished reading script.
    ===== PGPC_IT1_SBAE =====
done

Current directory: /home/postgres/pgpc-it1
Initialize GIN master
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
[INFO] target directory (/home/postgres/GHA/pgal/nodes/pbm) exists and not empty. Skip GIN initialization
postgreSQL01.0.0.HB's password:
Done
Start GIN master
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
postgreSQL01.0.0.HB's password:
Waiting for server to shut down... done
Server Stopped
server starting
postgreSQL01.0.0.HB's password:
initializing all the coordinator masters.
Initialize coordinator master shards
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "C.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory /home/postgres/GHA/pgal/nodes/coord_master1 ... ok
creating subdirectories ... ok
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting dynamic shared memory implementation ... posix
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... creating cluster information ... ok
syncing data to disk ... ok
freezing database template ... ok
freezing database template ... ok
freezing database postgres ... ok

WARNING: enabling "trust" authentication for local connections
you can change this by editing pg_hba.conf or using the options -A, -a, -auth-local and -auth-host, the next time you run initdb.
```

```
ALTER NODE coord1 WITH (HOST='10.0.0.182', PORT=30801);
ALTER NODE
CREATE NODE datanode_1 WITH (TYPE='datanode', HOST='10.0.0.183', PORT=40801, PRIMARY);
CREATE NODE
CREATE NODE datanode_2 WITH (TYPE='datanode', HOST='10.0.0.184', PORT=40802);
CREATE NODE
CREATE NODE datanode_3 WITH (TYPE='datanode', HOST='10.0.0.185', PORT=40803);
CREATE NODE
SELECT gpvc_pool_reload();
gpvc_pool_reload
-----
t
(1 row)

Dons.
EXECUTE DIRECT ON (datanode_1) 'CREATE NODE coord1 WITH (TYPE='\"'coordinator'\"', HOST='\"'10.0.0.182'\"', PORT=30801)';
EXECUTE DIRECT
EXECUTE DIRECT ON (datanode_1) 'ALTER NODE datanode_1 WITH (TYPE='\"'datanode'\"', HOST='\"'10.0.0.183'\"', PORT=40801, PRIMARY)';
EXECUTE DIRECT
EXECUTE DIRECT ON (datanode_1) 'CREATE NODE datanode_2 WITH (TYPE='\"'datanode'\"', HOST='\"'10.0.0.184'\"', PORT=40802)';
EXECUTE DIRECT
EXECUTE DIRECT ON (datanode_1) 'CREATE NODE datanode_3 WITH (TYPE='\"'datanode'\"', HOST='\"'10.0.0.185'\"', PORT=40803)';
EXECUTE DIRECT
EXECUTE DIRECT ON (datanode_1) 'SELECT gpvc_pool_reload()';
gpvc_pool_reload
-----
t
(1 row)

EXECUTE DIRECT ON (datanode_2) 'CREATE NODE coord1 WITH (TYPE='\"'coordinator'\"', HOST='\"'10.0.0.182'\"', PORT=30801)';
EXECUTE DIRECT
EXECUTE DIRECT ON (datanode_2) 'CREATE NODE datanode_1 WITH (TYPE='\"'datanode'\"', HOST='\"'10.0.0.183'\"', PORT=40801, PRIMARY)';
EXECUTE DIRECT
EXECUTE DIRECT ON (datanode_2) 'ALTER NODE datanode_2 WITH (TYPE='\"'datanode'\"', HOST='\"'10.0.0.184'\"', PORT=40802)';
EXECUTE DIRECT
```

4.3. Uruchomienie klastra:

```
PGXC start all
Start GTM master
server starting
Starting coordinator master.
Starting coordinator master coord1
2023-06-08 09:06:27.671 UTC [1038] LOG:  listening on IPv4 address "0.0.0.0", port 30001
2023-06-08 09:06:27.671 UTC [1038] LOG:  listening on IPv6 address "::", port 30001
2023-06-08 09:06:27.675 UTC [1038] LOG:  listening on Unix socket "/tmp/.s.PGSQL.30001"
2023-06-08 09:06:27.686 UTC [1038] LOG:  redirecting log output to logging collector process
2023-06-08 09:06:27.686 UTC [1038] HINT:  Future log output will appear in directory "pg_log".
Done.
Starting all the datanode masters.
Starting datanode master datanode_1.
Starting datanode master datanode_2.
Starting datanode master datanode_3.
2023-06-08 09:06:28.796 UTC [1039] LOG:  listening on IPv4 address "0.0.0.0", port 40001
2023-06-08 09:06:28.796 UTC [1039] LOG:  listening on IPv6 address "::", port 40001
2023-06-08 09:06:28.801 UTC [1039] LOG:  listening on Unix socket "/tmp/.s.PGSQL.40001"
2023-06-08 09:06:28.811 UTC [1039] LOG:  redirecting log output to logging collector process
2023-06-08 09:06:28.811 UTC [1039] HINT:  Future log output will appear in directory "pg_log".
2023-06-08 09:06:28.804 UTC [1036] LOG:  listening on IPv4 address "0.0.0.0", port 40002
2023-06-08 09:06:28.804 UTC [1036] LOG:  listening on IPv6 address "::", port 40002
2023-06-08 09:06:28.809 UTC [1036] LOG:  listening on Unix socket "/tmp/.s.PGSQL.40002"
2023-06-08 09:06:28.818 UTC [1036] LOG:  redirecting log output to logging collector process
2023-06-08 09:06:28.818 UTC [1036] HINT:  Future log output will appear in directory "pg_log".
2023-06-08 09:06:28.794 UTC [1037] LOG:  listening on IPv4 address "0.0.0.0", port 40003
2023-06-08 09:06:28.794 UTC [1037] LOG:  listening on IPv6 address "::", port 40003
2023-06-08 09:06:28.799 UTC [1037] LOG:  listening on Unix socket "/tmp/.s.PGSQL.40003"
2023-06-08 09:06:28.809 UTC [1037] LOG:  redirecting log output to logging collector process
2023-06-08 09:06:28.809 UTC [1037] HINT:  Future log output will appear in directory "pg_log".
Done.
```

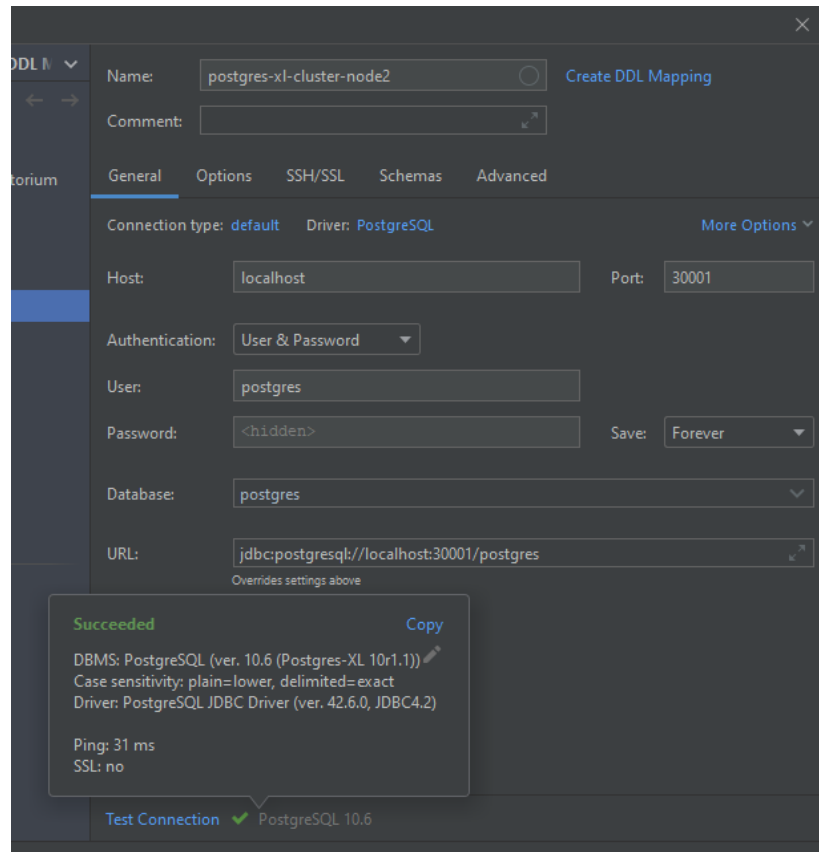
Sprawdzenie poprawności uruchomienia węzłów:

```
PGXC monitor all
Running: gtm master
Running: coordinator master coord1
Running: datanode master datanode_1
Running: datanode master datanode_2
Running: datanode master datanode_3
PGXC
```

node_name	node_type	node_port	node_host	nodeis_primary	nodeis_preferred	node_id
coord1	C	30001	10.0.0.102	f	f	1885696643
datanode_1	D	40001	10.0.0.103	t	f	-675012441
datanode_2	D	40002	10.0.0.104	f	f	-1047623914
datanode_3	D	40003	10.0.0.105	f	f	1787525382

(4 rows)

Sprawdzenie narzędziem DataGrip czy można podłączyć się do koordynatora:



Możemy teraz stworzyć testową tabelę w bazie danych łącząc się do węzła koordynatora.

Aby replikacja działała, podczas tworzenia tabeli należy określić w jaki sposób nastąpi rozproszenie tabeli na poszczególne węzły datanode.

1. Możemy rozproszyć tabelę za pomocą replikacji (DISTRIBUTE BY REPLICATION):

- Przy użyciu klauzuli DISTRIBUTE BY REPLICATION w poleceniu CREATE TABLE, tabela będzie replikowana na wszystkich węzłach Datanode w klastrze.
- Replikacja danych zapewnia wysoką dostępność, ponieważ każdy wiersz tabeli jest dostępny na każdym węźle Datanode.
- Wszelkie zmiany w danych będą automatycznie propagowane do wszystkich replik na węzłach Datanode.

2. Rozproszenie tabeli za pomocą partycjonowania (DISTRIBUTE BY HASH lub DISTRIBUTE BY ROUNDROBIN):

- Przy użyciu klauzuli DISTRIBUTE BY HASH, DISTRIBUTE BY ROUNDROBIN lub DISTRIBUTE BY MODULO w poleceniu CREATE TABLE, tabela zostanie podzielona i rozproszona między węzły Datanode na podstawie wartości klucza partycji (partition key).

- Klucz partycji może być jedną lub kilkoma kolumnami, a wybór między DISTRIBUTE BY HASH, a ROUNDROBIN czy MODULO zależy od preferowanego sposobu partycjonowania danych.
- **DISTRIBUTE BY HASH** używa funkcji haszującej do przyporządkowania wierszy do konkretnych węzłów Datanode
- **DISTRIBUTE BY ROUNDROBIN** równomiernie rozdziela wiersze między węzły
- **DISTRIBUTE BY MODULO** polega na przydzieleniu wierszy do węzłów Datanode na podstawie wyniku operacji modulo.

Ważne jest, aby pamiętać, że klauzula DISTRIBUTE BY odnosi się tylko do węzłów Datanode, a węzeł Coordinator jest odpowiedzialny za przetwarzanie zapytań i komunikację z węzłami Datanode w celu pobierania danych.

Stworzenie tabeli poprzez DISTRIBUTE BY REPLICATION:

```
✓ CREATE TABLE AGH_TEST_TAB
(
    COL1 INTEGER,
    COL2 INTEGER,
    COL3 TEXT,
    COL4 TEXT
) DISTRIBUTE BY REPLICATION;
```

Dodanie przykładowych danych:

```
INSERT INTO agh_test_tab (col1, col2, col3, col4)
VALUES
    (123, 456, 'Dell', 'Laptop'),
    (789, 101112, 'HP', 'Desktop'),
    (131415, 161718, 'Lenovo', 'All-in-One'),
    (192021, 222324, 'Apple', 'MacBook Pro'),
    (252627, 282930, 'Asus', 'Gaming Laptop'),
    (313233, 343536, 'Acer', 'Chromebook'),
    (373839, 404142, 'Microsoft', 'Surface Pro'),
    (434445, 464748, 'Sony', 'VAIO'),
    (495051, 525354, 'Toshiba', 'Satellite'),
    (555657, 585960, 'Samsung', 'Notebook');
```

Sprawdzenie czy dane dodały się, z węzła koordynatora (node2):

The screenshot shows a PostgreSQL console interface with two tabs: 'console_2 [postgres-xl-cluster-node2]' and 'console_1 [postgres-xl-cluster-node3]'. The active tab is 'console_2', which displays a successful query execution of `select * from agh_test_tab;`. Below the query, the 'Output' section shows the results of the query, displaying 10 rows of data from the `testdb.public.agh_test_tab` table. The data is presented in a table format with columns `col1`, `col2`, `col3`, and `col4`.

	col1	col2	col3	col4
1	123	456	Dell	Laptop
2	789	101112	HP	Desktop
3	131415	161718	Lenovo	All-in-One
4	192021	222324	Apple	MacBook Pro
5	252627	282930	Asus	Gaming Laptop
6	313233	343536	Acer	Chromebook
7	373839	404142	Microsoft	Surface Pro
8	434445	464748	Sony	VAIO
9	495051	525354	Toshiba	Satellite
10	555657	585960	Samsung	Notebook

Dane poprawnie zreplikowane na wszystkich węzłach datanode:

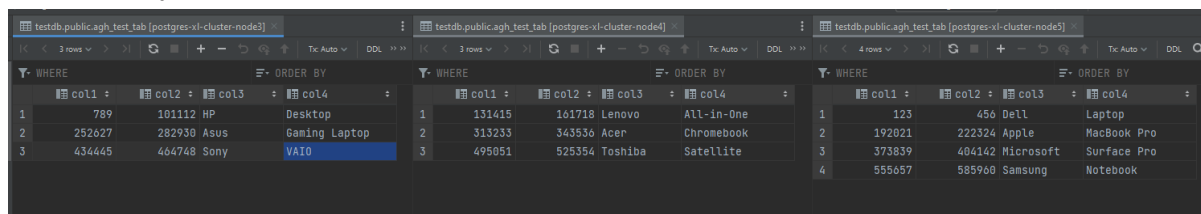
The screenshot displays three PostgreSQL console windows, each showing the same data from the `testdb.public.agh_test_tab` table. This confirms that the data has been successfully replicated across all data nodes in the cluster. The data is presented in a table format with columns `col1`, `col2`, `col3`, and `col4`.

	col1	col2	col3	col4
1	123	456	Dell	Laptop
2	789	101112	HP	Desktop
3	131415	161718	Lenovo	All-in-One
4	192021	222324	Apple	MacBook Pro
5	252627	282930	Asus	Gaming Laptop
6	313233	343536	Acer	Chromebook
7	373839	404142	Microsoft	Surface Pro
8	434445	464748	Sony	VAIO
9	495051	525354	Toshiba	Satellite
10	555657	585960	Samsung	Notebook

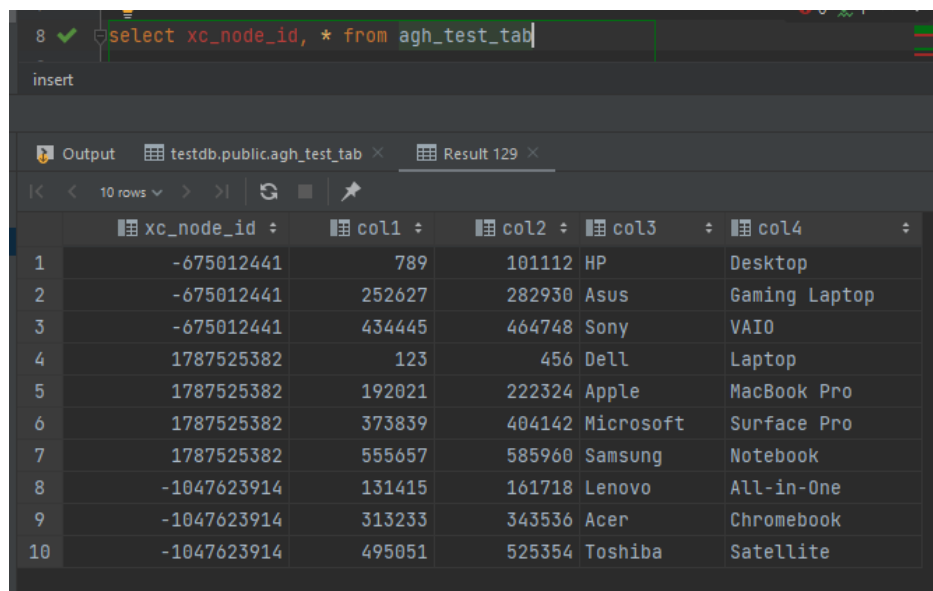
Dodanie tabeli ale rozproszenie danych poprzez partycjonowanie za pomocą algorytmu ROUNDROBIN:

```
CREATE TABLE AGH_TEST_TAB
(
    COL1 INTEGER,
    COL2 INTEGER,
    COL3 TEXT,
    COL4 TEXT
)
DISTRIBUTE BY ROUNDROBIN;
```

Dane zostały rozproszone:



col1	col2	col3	col4
789	101112	HP	Desktop
252627	282930	Asus	Gaming Laptop
434445	464748	Sony	VAIO



```
select xc_node_id, * from agh_test_tab
```

xc_node_id	col1	col2	col3	col4
-675012441	789	101112	HP	Desktop
-675012441	252627	282930	Asus	Gaming Laptop
-675012441	434445	464748	Sony	VAIO
1787525382	123	456	Dell	Laptop
1787525382	192021	222324	Apple	MacBook Pro
1787525382	373839	404142	Microsoft	Surface Pro
1787525382	555657	585960	Samsung	Notebook
-1047623914	131415	161718	Lenovo	All-in-One
-1047623914	313233	343536	Acer	Chromebook
-1047623914	495051	525354	Toshiba	Satellite

Czytamy w dokumentacji, że w przypadku klauzuli DISTRIBUTE BY HASH lub MODULO, należy określić kolumnę po której ma nastąpić partycjonowanie. Jeśli tego nie zrobimy, Postgres-XL będzie próbować znaleźć kolumnę, która ma deterministycznie rozpraszalny typ danych. Oznacza to, że Postgres-XL będzie szukać kolumny, której typ danych ma określoną logikę rozproszenia, umożliwiającą równomierne rozłożenie danych na różnych węzłach Datanode w klastrze.

Aby zapewnić wysoką dostępność (HA - High Availability) węzłów typu datanode, dodaliśmy dla poszczególnych węzłów master ich odpowiedniki typu slave i rozmieściliśmy na maszynach według następującej kolejności:

- 1.4 Datanode 1 (Master) - node3:
 - a. Datanode 1 (Slave) - node4
- 2.4 Datanode 2 (Master) - node4:
 - a. Datanode 2 (Slave) - node5
- 3.4 Datanode 3 (Master) - node5:
 - a. Datanode 3 (Slave) - node3

→ node3:

- ◆ Datanode 1 (Master)
- ◆ Datanode 3 (Slave)

→ node4:

- ◆ Datanode 2 (Master)
- ◆ Datanode 1 (Slave)

→ node5:

- ◆ Datanode 3 (Master)
- ◆ Datanode 2 (Slave)

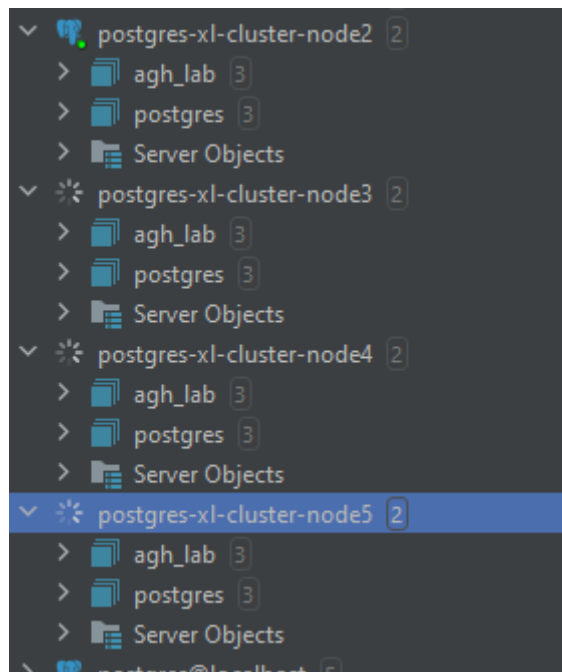
Konfiguracja:

```
#--- Slave -----
datanodeSlave=y      # Specify y if you configure at least one coordinator slave. Otherwise, the following
                     # configuration parameters will be set to empty values.
                     # If no effective server names are found (that is, every servers are specified as none),
                     # then datanodeSlave value will be set to n and all the following values will be set to
                     # empty values.

datanodeUserDefinedBackupSettings=n # Specify whether to update backup/recovery
                                   # settings during standby addition/removal.

datanodeSlaveServers=(node4 node5 node3) # value none means this slave is not available
datanodeSlavePorts=(40101 40102 40103) # Master and slave use the same port!
datanodeSlavePoolerPorts=(40111 40112 40113) # Master and slave use the same port!
datanodeSlaveSync=n # If datanode slave is connected in synchronized mode
datanodeSlaveDirs=($datanodeSlaveDir.1 $datanodeSlaveDir.2 $datanodeSlaveDir.3)
datanodeArchLogDirs=($datanodeArchLogDir.1 $datanodeArchLogDir.2 $datanodeArchLogDir.3)
```

Utworzyliśmy nową bazę danych “agh_lab” poprzez koordynatora, która natychmiast została zreplikowana na pozostałe węzły:



Stworzyliśmy nową tabelę:

```
1  ✓ CREATE TABLE agh_lab_tab
2  (
3      col1 SERIAL PRIMARY KEY,
4      col2 VARCHAR(50) not null,
5      col3 VARCHAR(50) not null,
6      col4 VARCHAR(50) not null
7  ) DISTRIBUTE BY REPLICATION;
8
```


Przykładowe dane:

```
insert into agh_lab_tab (col2, col3, col4)
values
  ('Harry Potter and the Philosopher's Stone', 'J.K. Rowling', 'Fantasy novel'),
  ('To Kill a Mockingbird', 'Harper Lee', 'Coming-of-age story'),
  ('Pride and Prejudice', 'Jane Austen', 'Romance novel'),
  ('The Great Gatsby', 'F. Scott Fitzgerald', 'Historical fiction'),
  ('1984', 'George Orwell', 'Dystopian novel'),
  ('The Catcher in the Rye', 'J.D. Salinger', 'Coming-of-age story'),
  ('Moby-Dick', 'Herman Melville', 'Adventure novel'),
  ('The Lord of the Rings', 'J.R.R. Tolkien', 'Fantasy novel'),
  ('To the Lighthouse', 'Virginia Woolf', 'Modernist novel'),
  ('The Odyssey', 'Homer', 'Epic poem');
```

UWAGA!!!

Klaster do tego momentu działał poprawnie jeśli chodzi o replikację jak propagację danych pomiędzy węzły “datanode”, ale niestety najnowsza dostępna wersja **postgres-xl-10r1.1** nie działała u nas jeśli chodzi failover, czyli przełączenie na datanode typu slave, jeśli jego master nie był dostępny. Od tego momentu, po wielu nieudanych próbach uruchomienia poprawnie “failover”, postanowiliśmy postawić klaster na starszej stabilnej wersji **postgres-xl-9.5r1.6**

Postgres-xl nie obsługuje automatycznego przełączania podczas gdy jeden node jest niedostępny. Sprawdzenie czy ręczny failover zadziała. Wyłączenie jednej maszyny datanode:

```
PGXC monitor all
Running: gtm master
Running: coordinator master coord1
Running: datanode master datanode_1
Running: datanode slave datanode_1
Running: datanode master datanode_2
Running: datanode slave datanode_2
Running: datanode master datanode_3
Running: datanode slave datanode_3
PGXC stop -m immediate datanode master datanode_1
Stopping datanode master datanode_1.
Done.
PGXC monitor all
Running: gtm master
Running: coordinator master coord1
Not running: datanode master datanode_1
Running: datanode slave datanode_1
Running: datanode master datanode_2
Running: datanode slave datanode_2
Running: datanode master datanode_3
Running: datanode slave datanode_3
```

4.4 Testowanie klastra:

Test Failover:

Uruchomienie poleceniem “failover” maszyny slave, tak aby przejął kontrolę:

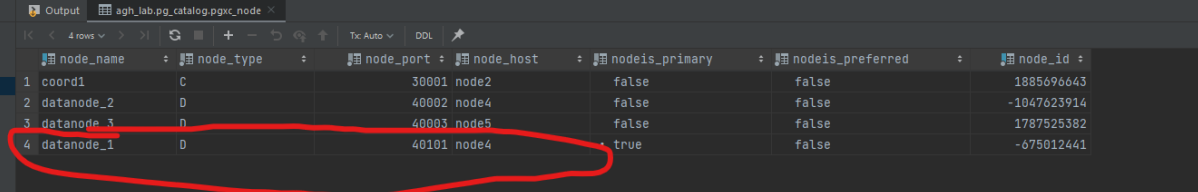
```
PGXC failover datanode datanode_1
Failover specified datanodes.
Failover the datanode datanode_1.
Failover datanode datanode_1 using GTM itself
Actual Command: ssh postgres@node4 "( pg_ctl promote -Z datanode -D /home/postgres/DATA/pgxl/nodes/dn_slave.1 ) > /tmp/node1_STDOUT_25100_125 2>&1" < /dev/null > /dev/null 2>&1
Bring remote stdout: scp postgres@node4:/tmp/node1_STDOUT_25100_125 /tmp/STDOUT_25100_126 > /dev/null 2>&1
Actual Command: ssh postgres@node4 "( pg_ctl restart -w -Z datanode -D /home/postgres/DATA/pgxl/nodes/dn_slave.1 -o -i; sleep 1 ) > /tmp/node1_STDOUT_25100_127 2>&1" < /dev/null > /dev/null 2>&1
Bring remote stdout: scp postgres@node4:/tmp/node1_STDOUT_25100_127 /tmp/STDOUT_25100_128 > /dev/null 2>&1
2023-06-11 17:40:22.075 CEST [17076] LOG:  listening on IPv4 address "0.0.0.0", port 40101
2023-06-11 17:40:22.075 CEST [17076] LOG:  listening on IPv6 address ":::", port 40101
2023-06-11 17:40:22.079 CEST [17076] LOG:  listening on Unix socket "/tmp/.s.PGSQL.40101"
2023-06-11 17:40:22.093 CEST [17076] LOG:  redirecting log output to logging collector process
2023-06-11 17:40:22.093 CEST [17076] HINT:  Future log output will appear in directory "pg_log".
ALTER NODE
pgxc_pool_reload
-----
t
(1 row)

EXECUTE DIRECT
pgxc_pool_reload
-----
t
(1 row)

EXECUTE DIRECT
pgxc_pool_reload
-----
t
(1 row)

EXECUTE DIRECT
pgxc_pool_reload
-----
t
(1 row)

Done.
```



	node_name	node_type	node_port	node_host	node_is_primary	node_is_preferred	node_id
1	coord1	C	30001	node2	false	false	1885696643
2	datanode_2	D	40002	node4	false	false	-1047623914
3	datanode_3	D	40003	node5	false	false	1787525382
4	datanode_1	D	40101	node4	true	false	-675012441

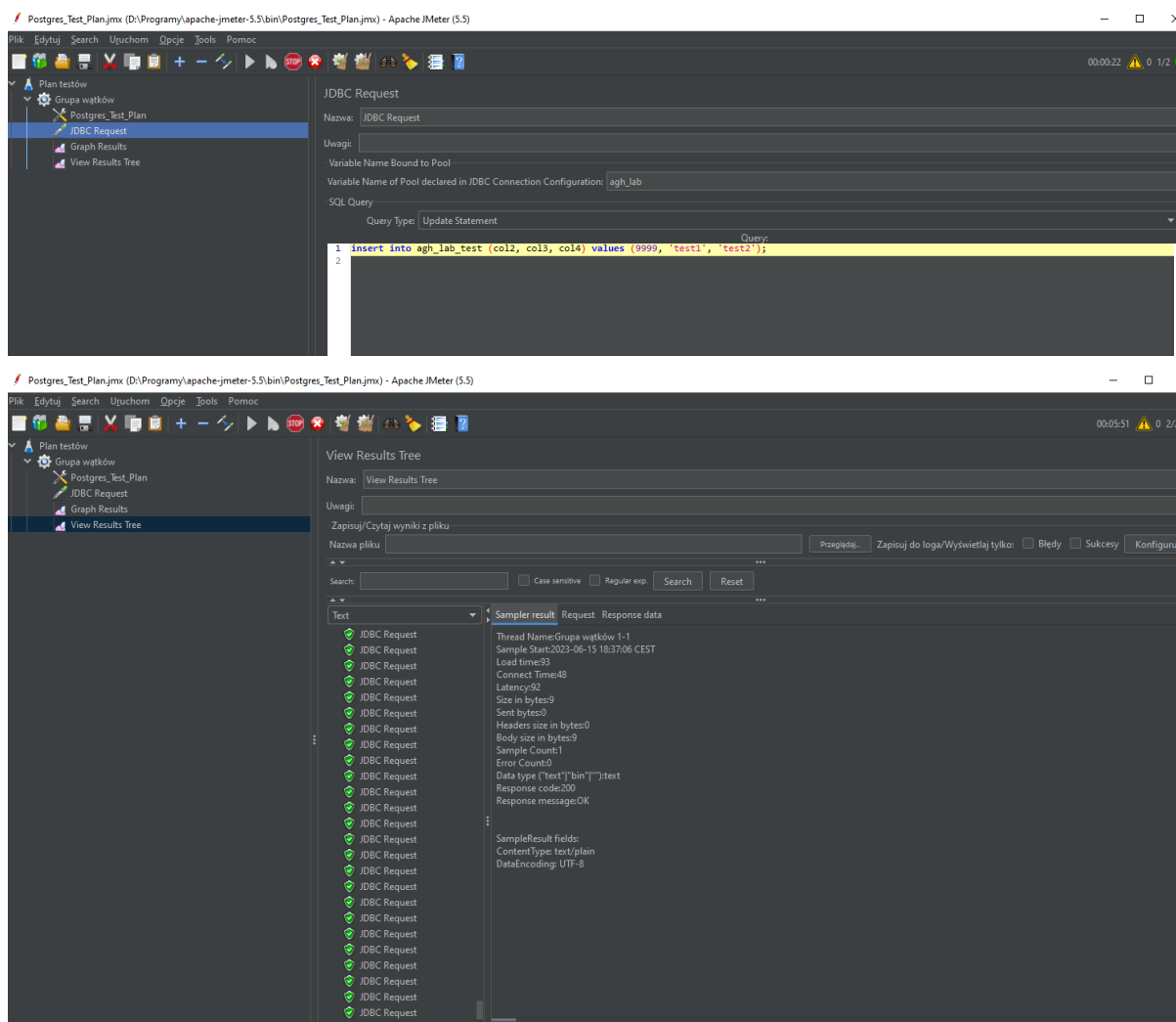
```
PGXC monitor all
Running: gtm master
Running: coordinator master coord1
Running: datanode master datanode_1
Running: datanode master datanode_2
Running: datanode slave datanode_2
Running: datanode master datanode_3
Running: datanode slave datanode_3
```

Podczas wykonania polecenia ręcznego failoveru, konfiguracja pgxc_ctl.conf zmieniła się następująco:

```
#=====
Updated due to the datanode failover, datanode_1, 20230611_17:40:23
datanodeMasterServers=( node4 node4 node5 )
datanodePorts=( 40101 40002 40003 )
datanodePoolerPorts=( 40111 40012 40013 )
datanodeMasterDirs=( /home/postgres/DATA/pgxl/nodes/dn_slave.1 /home/postgres/DATA/pgxl/nodes/dn_master.2 /home/postgres/DATA/pgxl/nodes/dn_master.3 )
datanodeSlaveServers=( none node5 node3 )
datanodeSlavePorts=( none 40102 40103 )
datanodeSlavePoolerPorts=( none 40112 40113 )
datanodeSlaveDirs=( none /home/postgres/DATA/pgxl/nodes/dn_slave.2 /home/postgres/DATA/pgxl/nodes/dn_slave.3 )
# End of the update
```

Testowanie wydajności:

Aby przetestować klastr pod względem wydajności zdecydowaliśmy się wykorzystać narzędzie Apache JMeter. Przeprowadziliśmy serię testów obciążeniowych, które miały na celu ocenę wydajności naszego klastra. Dzięki JMeterowi mogliśmy generować duże obciążenie, symulując równoczesne żądania od wielu użytkowników, oraz monitorować parametry wydajnościowe, takie jak czasy odpowiedzi, przepustowość oraz obciążenie serwera.













4.5 Monitorowanie klastra:

Monitorowanie klastra za pomocą Zabbix:

Aby monitorować nasz klaster Postgres-XL, zdecydowaliśmy się na wykorzystanie oprogramowania Zabbix. Poniżej przedstawiamy kroki, które podjęliśmy w celu skonfigurowania monitoringu:

a) Instalacja oprogramowania Zabbix poprzez zdefiniowanie jego składowych w pliku docker-compose.yml w celu uruchomienia oprogramowania jako kontenery Docker.

```
docker-compose.yml > {} services > {} zabbix-web > {} networks
1 version: "3"
2 services:
3   zabbix-server:
4     image: zabbix/zabbix-server-mysql:latest
5     container_name: zabbix-server
6     restart: always
7     ports:
8       - 10051:10051
9     environment:
10      - DB_SERVER_HOST=mysql-server
11      - MYSQL_DATABASE=zabbix
12      - MYSQL_USER=zabbix
13      - MYSQL_PASSWORD=zabbix
14      - MYSQL_ROOT_PASSWORD=zabbix
15     depends_on:
16       - mysql-server
17     networks:
18       - zabbix-net
19
20   zabbix-web:
21     image: zabbix/zabbix-web-nginx-mysql:latest
22     container_name: zabbix-web
23     restart: always
24     ports:
25       - 80:8080
26     environment:
27      - DB_SERVER_HOST=mysql-server
28      - MYSQL_DATABASE=zabbix
29      - MYSQL_USER=zabbix
30      - MYSQL_PASSWORD=zabbix
31      - MYSQL_ROOT_PASSWORD=zabbix
32      - ZBX_SERVER_HOST=zabbix-server
33      - PHP_TZ=Europe/Warsaw
34     depends_on:
35       - zabbix-server
36     networks:
37       - zabbix-net
38
39   mysql-server:
40     image: mysql:latest
41     container_name: mysql-server
42     restart: always
43     ports:
44       - 3306:3306
45     environment:
46      - MYSQL_DATABASE=zabbix
47      - MYSQL_USER=zabbix
48      - MYSQL_PASSWORD=zabbix
49      - MYSQL_ROOT_PASSWORD=zabbix
50     volumes:
51       - mysql-data:/var/lib/mysql
52     networks:
53       - zabbix-net
54
55 networks:
56   zabbix-net:
57
58 volumes:
59   mysql-data:
```

<input type="checkbox"/>	Name	Image	Status	Port(s)
<input type="checkbox"/>	 postgres-xl-cluster	-	Running (3/3)	
<input type="checkbox"/>	 zabbix-web cad77c71ab88 	zabbix/zabbix-web-nginx-m	Running	80:8080 
<input type="checkbox"/>	 zabbix-server 80c22483b70f 	zabbix/zabbix-server-mysql	Running	10051:10051 
<input type="checkbox"/>	 mysql-server bbdebe850a12 	mysql:latest	Running	3306:3306 

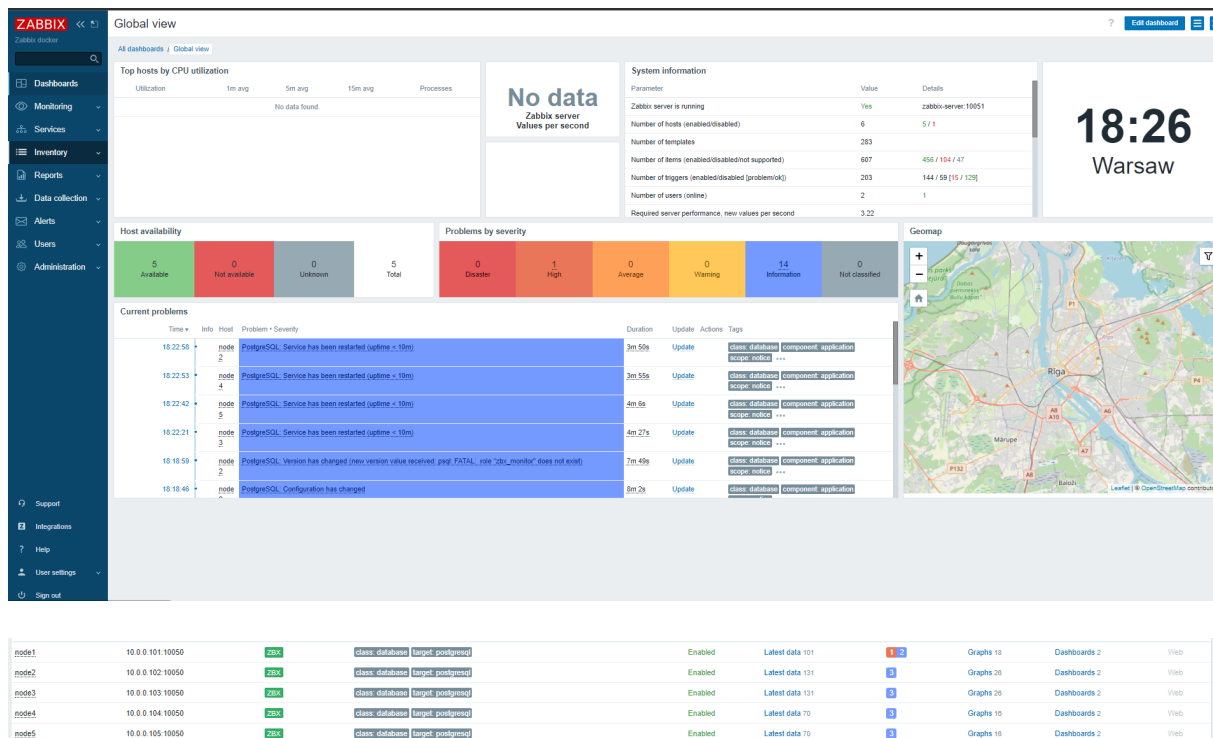
b) Skonfigurowanie agentów Zabbix na każdym węźle klastra Postgres-XL.
Zautomatyzowanie poprzez wykorzystanie playbooków Ansible:

```

1  ---
2  ✓ - name: Install zabbix agent on remote hosts
3    hosts: target_hosts
4    remote_user: postgres
5    tasks:
6      - name: Install zabbix agent
7        shell: apt install -y zabbix-agent
8        become: true
9        remote_user: postgres
10       become_exe: /usr/bin/sudo
11       become_method: sudo
12
13     ✓ - name: Copy file to remote host
14       copy:
15         src: zabbix_agentd.conf
16         dest: /etc/zabbix/zabbix_agentd.conf
17       become: true
18       remote_user: postgres
19       become_exe: /usr/bin/sudo
20       become_method: sudo
21
22     ✓ - name: Restart zabbix agent
23       shell: systemctl restart zabbix-agent
24       become: true
25       remote_user: postgres
26       become_exe: /usr/bin/sudo
27       become_method: sudo
28
29     ✓ - name: Restart host
30       shell: reboot
31       become: true
32       remote_user: postgres
33       become_exe: /usr/bin/sudo
34       become_method: sudo
35

```

c) Konfiguracja parametrów monitorowania, takich jak obciążenie systemu, zużycie zasobów, statystyki bazy danych i sieci.



ZABBIX

zabbix doctor

Dashboard

Monitoring

Problems

Hosts

Latest data

Maps

Discovery

Services

Inventory

Reports

Data collection

Alerts

Users

Administration

Support

Integrations

Help

User settings

Sign out



ZABBIX

zabbix doctor

Dashboard

Monitoring

Problems

Hosts

Latest data

Maps

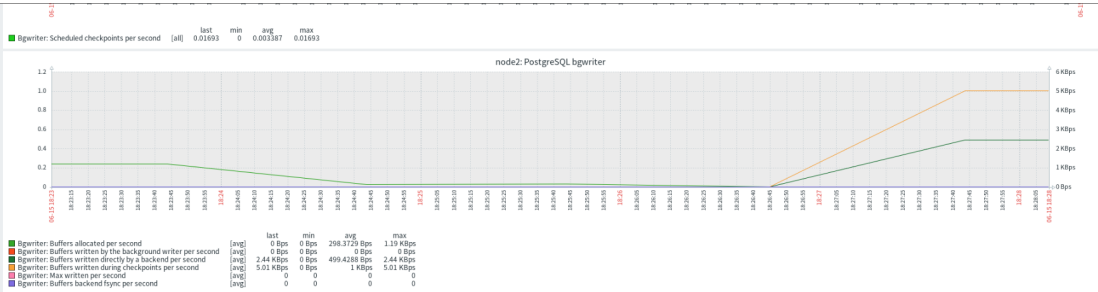
Discovery

Services

Inventory

Reports

Data collection



Monitorowanie klastra za pomocą Grafana:

Aby wizualizować zebrane dane monitorowania, zaimplementowaliśmy narzędzie Grafana. Oto kroki, które podjęliśmy w celu skonfigurowania wizualizacji:

1. Uruchomienie darmowej wersji Grafana Cloud.
2. Konfigurowanie źródła danych w Grafanie, które pobiera dane przekazane przez Grafana Agent, zainstalowanego na maszynie klastra.
3. Utworzenie paneli i wykresów w Grafanie, aby wizualnie przedstawić dane monitorowania klastra Postgres-XL.

